Performability Monitoring and Modelling of *ANSA*ware Environments

Raymond H. Pijpers†, Leonard J.N. Franken†
Boudewijn R.H.M. Haverkort‡

†PTT Research
P.O. Box 15000, 9700 CD Groningen, The Netherlands
E-mail: r.h.pijpers@research.ptt.nl, l.j.n.franken@research.ptt.nl
‡University of Twente
Department of Computer Science Tele-Informatics and Open Systems
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: b.r.h.m.haverkort@cs.utwente.nl

February 28, 1994

Abstract

The Performability Manager (PM) is a distributed system component which maintains the application-requested Quality of Service (QoS) by dynamically reconfiguring *ANSA*ware-based distributed applications, using a model-based optimization procedure. The PM receives information about the *ANSA*ware-based application from a distributed monitoring process based on JEWEL and DEMON. With this information, and using predefined stochastic Petri net (SPN) models of *ANSA*ware applications, the PM automatically constructs an overall SPN performability model which is subsequently used for the determination of the provided QoS. Based on the analysis results, the PM can decide to initiate on-line system reconfigurations, if needed to maintain the requested QoS. *ANSA*ware provides facilities for these dynamic reconfigurations.

In this paper we focus on monitoring the *ANSA*ware-based experimental distributed environment in which the modelling and evaluation aspects are totally automated. We also show the feasibility of the proposed PM by presenting some operational results. The contents of this paper is a summary of the *ANSA*ware related topics in "Modelling Aspects of Model Based Dynamic QoS Management by the Performability Manager" by Franken, Haverkort and Pijpers and the M.Sc. thesis with the same title as this paper by Pijpers.

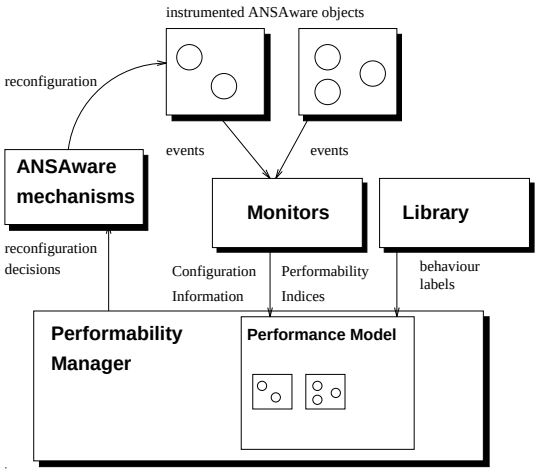


Figure 1: The Performability Manager in an *A*NSAware environment

1 Introduction

For modern distributed systems it is important to be able to realize and maintain a requested Quality of Service (QoS). The QoS can degrade for several reasons: the addition of new applications, the updating of applications, the change of workload (new users) or the occurrence of failures and repairs.

The *Performability Manager* (PM) is a distributed system component which maintains the application-requested QoS by dynamically reconfiguring a distributed system [4].

In a nutshell the PM can be described as a distributed system feedback manager (see Figure 1). If the QoS decreases, it is detected by the monitoring tool and notified to the PM. The PM creates dynamically new alternative configurations and calculates the QoS of these configurations using the Performability Model. Together with the monitoring results and the behaviour labels (SPN sub-models) the configuration can be transformed into an SPN model which describes the dynamical behaviour of the configuration. The current configuration will be updated to obtain the best alternative configuration calculated. The PM uses the *A*NSAware mechanisms to perform these reconfigurations. After reconfiguration the cycle starts again; the monitors will monitor the object system and notify any decrease of QoS to the PM[1].

This paper is further organized as follows. In Section 2 the experimental

[1] For all specific information on the Performability Manager is referred to the IEEE paper [4] and proceedings paper [5]. In this paper the monitoring and modelling of *A*NSAware environments is addressed as presented in [5] and fully documented in the M.Sc. thesis [11].

environment and the *A*NSAware computational model of our application is presented. In Section 3 the creation of a performability model using predefined stochastic Petri net models of the system components is presented. The monitoring of the experimental distributed environment is discussed in Section 4 and parameterization and first results on measurements are presented in Section 5. Specific *A*NSAware related topics of performability monitoring and modelling is discussed in Section 6. Finally, in Section 7, the implementation and operational issues of the presented modelling techniques, our ongoing research and outlines for future research are discussed.

2 An *A*NSAware-based distributed environment

The application in our distributed environment is realized using *A*NSAware. From the ANSA computational model we want to come to a *performability model*, which will be discussed in later sections.

In Section 2.1 we present a distributed application which is used as an experimental application. Section 2.2 presents the experimental application using the *A*NSAware computational model.

2.1 An *A*NSAware-based number translation service

In this section we describe the (telephone) number translation service (NTS) as provided in intelligent networks [2, 6, 12]. In the sequel we will refer to this application as the IN/ANSA application.

End-Users are submitting requests or tasks for the application at a certain rate. Since we do not have real users, we simulate the user behaviour by a so-called *Generating Component* (GC). The GC generates the calls for the NTS.

The NTS is provided by the following application components (see also Figure 2):

1. *The Selection Component* (SC): this component selects a service using the contents of the requests it receives (number translation service in this example).

2. *The Number Translation Component* (NTC): this component receives requests for number translations. The NTC sends a request to a database component for the required number and to a billing component for the creation of a bill. The number received from the database is returned to the SC.

3. *The DataBase Component* (DBC): this component receives requests for specific numbers. It will fetch the number from disk and return the number to the component which requested the number.

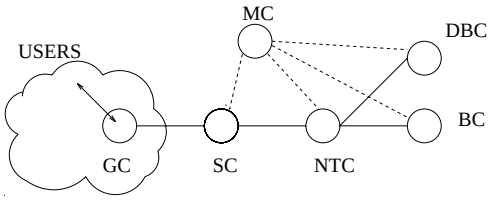


Figure 2: The experimental *A*NSAware application

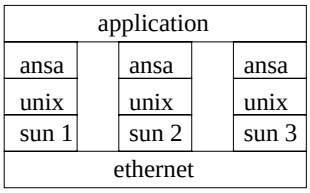


Figure 3: The distributed system

4. *The Billing Component* (BC): this billing component receives requests for the preparation of a billing record.

The *Management Component* (MC) does not belong specifically to the NTS, but provides the PM with the necessary "buttons to push" to perform a reconfiguration. The MC in turn uses *A*NSAware facilities to perform necessary reconfigurations. The other components (SC, NTC, DBC and BC) are components of the application and can be controlled by the MC.

For the experimental application we use a small distributed system consisting of three SUN SPARC workstations connected by an Ethernet as depicted in Figure 3. The workstations run UNIX and, on top of that, *A*NSAware. Of course, more heterogeneous environments are possible as well, e.g. using both SUNs and PCs. Within this experimental distributed environment we use two monitors, DEMON and JEWEL. DEMON, the Distributed Environment MONitor [10], is used to visualize the structure of the experimental distributed environment. JEWEL [9] is used to do performance measurements in the experimental distributed environment.

2.2 The computational model of IN/*A*NSA

In our experimental distributed environment the computational objects are the application components of the distributed system. One or more computational

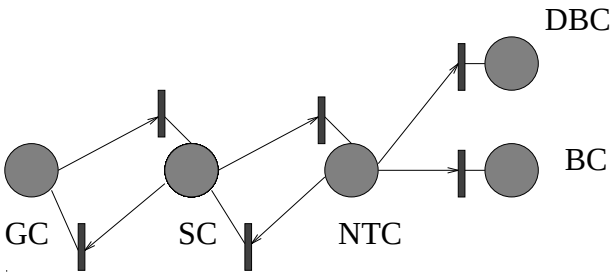


Figure 4: The experimental application described in a computational form

objects make up a distributed application as shown in Figure 4. Each computational object has been implemented as a capsule. All invocations for the experimental application are announcements, except for those between the NTC and the BC and those between the NTC and the DBC; these are interrogations.

3 A performability model of INA/ANSA

To evaluate an alternative configuration we need a performability model. Therefore, an alternative configuration is transformed into a performability model by replacing each component by a predefined stochastic Petri net (SPN) model. The resulting performability models are both flexible and relatively easy to solve by current day software tools [3, 7]. In this paper we will deal with the performance aspects of the model only.

In Section 3.1 we present the generic SPN modelling of user, application and system components. The performability model of the experimental distributed environment is presented in Section 3.2.

3.1 The SPN models used to realize the performability model

In this section we present a generic way to transform each component of the distributed environment into an SPN sub-model. We start with the application level, then the system level and finally present how the users are modelled using SPN.

For each operation or service provided by an application component an SPN model component is predefined. In such an SPN model a service is represented by a timed transition. The invocation of a service at the interface by a client is represented by putting a token in the corresponding "service-input place". Resources must be allocated (e.g. an CPU) and the operation can be performed

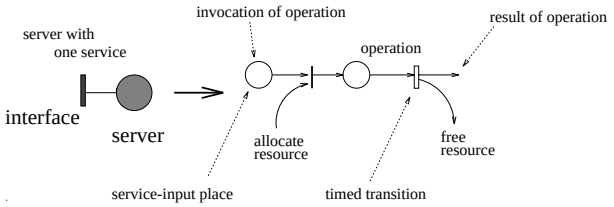


Figure 5: The SPN representation of an ANSAware service provision

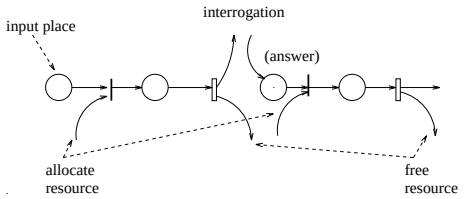


Figure 6: The SPN representation of an announcement or interrogation operation

(the timed transition). In Figure 5 we see (at the right hand side of the arrow) the SPN representation of one operation of a computational object (shown at the left hand side of the arrow) or application component. The output of the timed transition, i.e. the operation, is an announcement or an interrogation to another operation (see Figure 6). With an interrogation invocation as output the component will await for an answer and then continue operation.
The duration of an operation is represented by a timed transition. These transitions represent the work demanded from the resource, for example the CPU busy time. We can estimate these parameters by running and monitoring the component in isolation (one component on a single workstation).

The communication between components can be represented in a similar way. For each (remote) operation, or communication between two application components allocated to different system parts, a network link must be allocated. The duration of a communication operation is also represented by a timed transition. In this case a timed transition represents the communication time per invocation of an operation per network link (see [4]).

The generation of requests by the USERS is modelled as a Poisson arrival process, represented by a single timed transition.

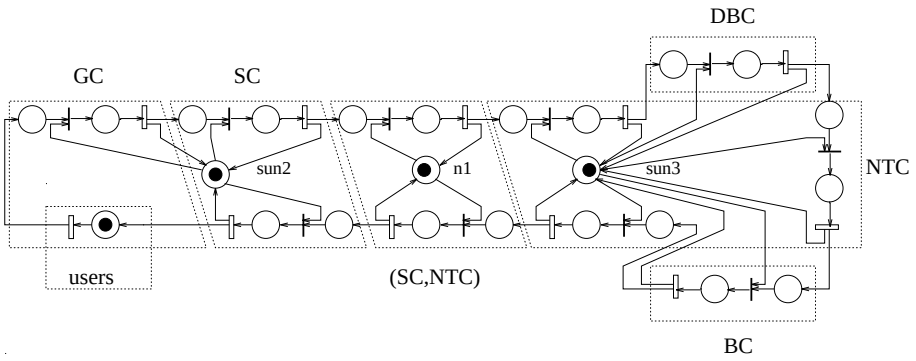


Figure 7: The SPN model of configuration M^1

3.2 The performability model of the IN/ANSA environment

Tools for SPN analysis normally only allow finite state space models. This does not correspond to the experimental environment. However, we can approximate an open model by designing a closed model with a large customer population. The average *request rate* is modelled by the component USERS. In this paper three configurations of the application are taken as example: configurations M^1, M^2 and M^3, with each their own allocation of components. M^1 features one NTS service of which the components are distributed on two SUNs. M^2 features one NTS service of which the components are allocated on one SUN and M^3 features *two* NTS service of which the components are distributed on two SUNs. In Figure 7 the SPN representation of the distributed environment, using configuration M^1, is given using the predefined SPN models.

4 Monitoring of ANSAware applications

Two different monitoring tools, the DEMON and the JEWEL tool, monitor the experimental environment introduced in Section 2. The DEMON tool [10] monitors and visualizes the functional behaviour and configuration of the ANSAware components on the system nodes. These can be used to provide the performability model with configuration information. The JEWEL monitoring tool [9] extracts performability indices from the ANSAware environment and visualizes them for each component on a graphical display. The performability indices are used to detect a decrease of QoS and to parameterize the performability model.

In order to provide the monitoring tools with the information needed, the ANSAware application components have to be instrumented with additional code for both monitoring systems. Instrumentation for the DEMON tool is performed

automatically by a pre-compiler designed and implemented at PTT Research [8]. Instrumentation for the JEWEL monitoring tool is performed in a generic manner using the *A*NSAware operations as a reference point to detect relevant events. The implementation of the invocation of an operation is embraced by the two events: *request* and *confirm*. These events are detected by JEWEL and used to derive the turnaround time of an operation. The implementation of the operation is also embraced by two events: *indication* and *response*. These are detected by JEWEL and used to derive the service time of an operation, as shown in Figure 9. A detailed prescription of generic instrumentation for *A*NSAware is provided in [11].

5 Experiences with monitoring, modelling and evaluation

A performability model of a distributed application can automatically be constructed guided by three input sources (see also Figure 8):

1. *A library of predefined SPN models.* For each *A*NSAware and system component a model has to be available in a library.

2. *Configuration determination.* The configuration has to be obtained from the system to construct the model from the predefined model components in the library. The DEMON monitor provides this configuration information.

3. *Performability indices determination.* We use the performability monitoring measurements provided by JEWEL to determine the transition rates of the timed transitions in the SPN.

The method of tuning the performability model [4], obtains the transition rates for the SPN from the requirements of the components and the capacities of the system nodes. A major drawback of this method is the required *a priori* determination of the requirements and capacities. Because the source code of the *A*NSAware components is processed by pre-compilers and linked with library functions, exact requirements of the components with respect to processing workload, communication workload, memory access, etc. are hard to assess. Capacities of the system nodes may be exactly specified by the manufacturers, but mechanisms like memory caching or disk access cause dynamically changing capacities of the system nodes. Therefore, we have used a more practical approach to parameterize the performability model, guided by the measurements provided by the JEWEL monitoring tool.

The transition rates can be obtained by measuring the service times of the individual components. In Figure 9 a timing diagram is depicted containing the monitored time-stamps of the events: request, confirm, indication and response.

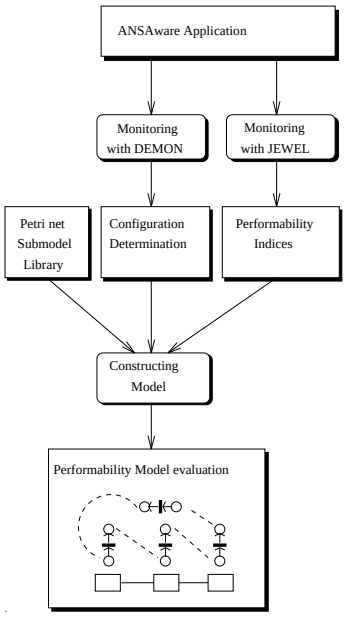


Figure 8: The performability model is constructed from three input sources

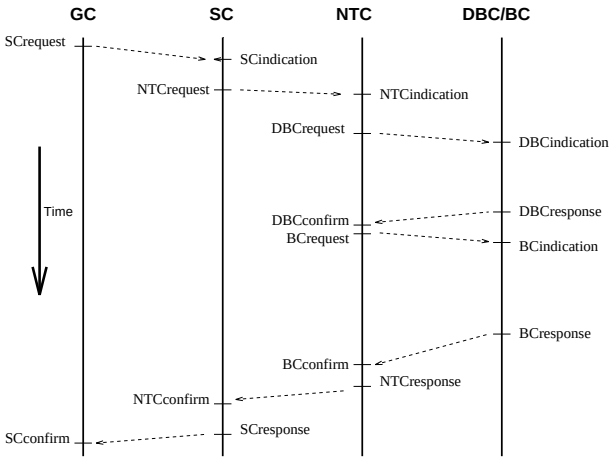


Figure 9: The timing diagram for one configuration

The service times can be derived from these measurements under minimal load. No queueing will occur under minimal load, so the residence time of a component will be equal to the service time of that component decreased by the residence times of the interrogation operations invoked during the service provisioning and the encountered communication delays:

$$T_j = \frac{1}{R_j - \sum_{i \in K} R_i - \sum_{i \in K} C_i}$$

where T_j is the transition rate of the sub-model of component j, R_j is the average turnaround time of component j and K is the set of operations invoked (as an interrogation) by component j. R_i is the average turnaround time of operation i and C_i the average communication delay to component i.

As an example consider the service time of the NTC in Figure 9. The service time of component NTC can be derived from the residence time of NTC (NTCresponse - NTCindication) minus the residence times of DBC and BC and the communication delays (differences between request and indication and the differences between response and confirm). In this way service times and communication delays can be derived from the measurements depicted in the diagram. A drawback of this method is that for each combination of components and system nodes a measurement under minimal load has to be done to obtain the residence time without queueing. A major advantage of this method, however, is the higher level of abstraction maintained, i.e. the capacities of the system nodes and the requirements of the components are implicitly incorporated.

We now discuss some comparative results from the modelling and monitoring. The IN*A*NSA application has been monitored using different alternative configurations. The performability model has been parameterized with statistics (averages) over the measurements, obtained by monitoring the different configurations under minimal load. This leads to one set of parameters applicable for all configurations. The SPNP (stochastic Petri net package) implementation [3] of the performability model has been verified with the performability indices actually measured by the JEWEL monitor under various workloads.

In Table 1 the monitored results for the three different configurations are presented in comparison with the values calculated by SPNP. We see that the model results, under minimal load, come very close to the values actually measured. Notice that these results are obtained using a very simple performance model, only taking into account application components and CPU possession.

Finally, we compare the measured results with the model evaluation results under higher load. Note that the models parameterization is the same as for the minimal load case. The configurations investigated are M^1, M^2 and M^3. Due to scheduling strategies of *A*NSAware the approximations for the turnaround time of the "internal" components, i.e. NTC, SC, BC and DBC are not comparable

config.		Time in ms. Monitored	SPNP	Difference
M^1	Turnaround time of GC	112	113	+0.9%
	Turnaround time of SC	91	88	-3.4%
	Turnaround time of NTC sun3	76	73	-4.1%
M^2	Turnaround time of GC	157	157	0%
	Turnaround time of SC	132	132	0%
	Turnaround time of NTC sun2	103	103	0%
M^3	Turnaround time of GC	138	135	-2.2%
	Turnaround time of SC	115	110	-4.5%
	Turnaround time of NTC sun2	106	103	-2.9%
	Turnaround time of NTC sun3	73	73	0%

Table 1: Evaluation SPNP model with monitoring results under minimal load for configurations M^1, M^2 and M^3.

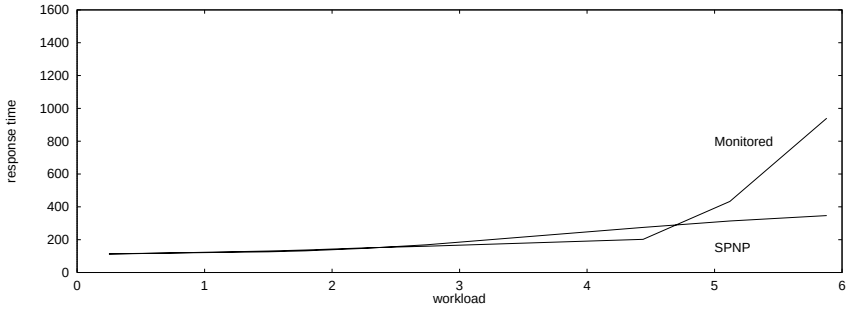
to the SPNP results. More important for the performance, however, is the QoS provided to the user, i.e. the turnaround time of the complete application, which is equal to the turnaround time of GC. We therefore address this metric.

The results of the SPNP model and the measurements are graphically depicted in Figure 10. For each configuration eight monitoring sessions were conducted for different workloads (λ). The results of the SPNP model are reasonably good (less than 10% error) when the load is low to moderate. When the load increases, however, the monitoring results differ substantially from the results calculated by SPNP. The workload range of our interest is the moderate range were the turnaround time does not exceed the requested QoS. If the QoS is violated (or the turnaround time has increased significantly) the performability manager is triggered and runs the Performability Model for alternative configurations. Further research is necessary to estimate the level of confidence we can put in our models.

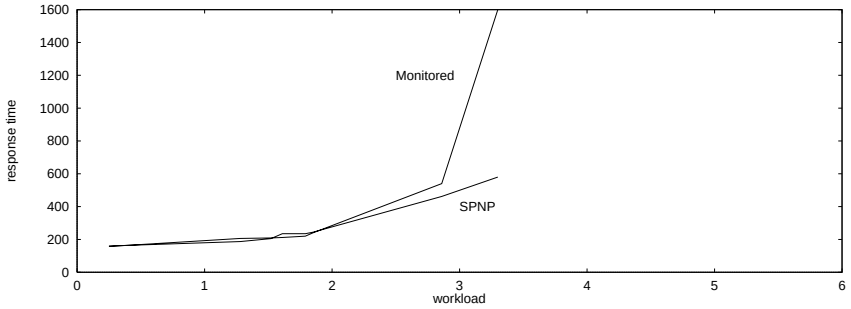
6 *A*NSAware related topics

In the previous sections the monitoring, modelling and the evaluating of *A*NSAware environments has been described. In this section we will focus on the specific *A*NSAware related topics of these activities.

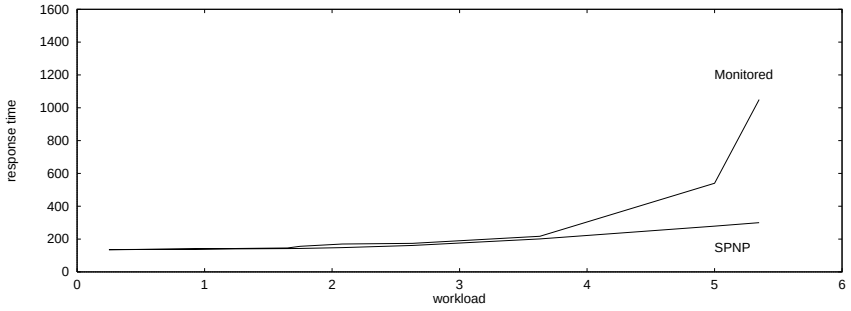
When monitoring an *A*NSAware application the RPC mechanism has shown to be a good framework for instrumentation of the source code. With the four events identified the major part of the relevant performability indices can be calculated. The adjustments to the *A*NSAware source code are few and well



Configuration M^1



Configuration M^2



Configuration M^3

Figure 10: Evaluation SPNP model with monitoring results for configurations M^1, M^2 and M^3.

prescribed to perform monitoring with Jewel. A drawback of *A*NSAware is the lack of global time, which causes that the communication delays can not be calculated exactly.

The computational model has been used as a framework to perform performability modelling. The *A*NSAware computational objects can be described by stochastic Petri net sub-models. Guided by the configuration of the objects and the monitoring results with these sub-models an overall model is composed and evaluated. The results of these evaluations are reasonably good when workload is low to moderate. When the load increases further the turnaround time of the application increases explosively. This phenomenon can be explained by the scheduling strategy of *A*NSAware. The submission and reception of number translations are both implemented in one capsule (the GC). When the load increases the submission side of the capsule takes over the CPU possession[2] at expense of the reception side, which causes the increase of the turnaround time.

Another problem encountered during the monitoring of the application was the implementation of the *announcement* operations. The announcements seem to get lost when load increases, even when capsules were running on *one* machine and enough tasks and threads were allocated. The solution to this problem was to simulate announcements with interrogations, using the `voucher/redeem` construct[3].

Concluding we can state that the *A*NSAware platform provides a good framework to do performability monitoring and modelling using generic prescriptions.

7 Discussion and future work

In this paper we focused on the monitoring and modelling aspects of *A*NSAware environments. We presented an *A*NSAware-based experimental distributed environment in which the modelling and evaluation aspects are totally automated. We proposed a generic modelling strategy in which the structure of the client/server and the computational model of the *A*NSAware computing platform are used. This structure allows for a generic transformation of the computational models into performability models using predefined SPN models.

The PM receives information about the *A*NSAware-based application from a distributed monitoring process based on JEWEL and DEMON. With this information, and the SPN model library of *A*NSAware applications, the PM automatically constructs an overall SPN performability model which is subsequently used for the determination of the provided Quality of Service (QoS).

Currently we are working on proper mapping algorithms for the creation of the alternative configurations.

The required on-line and therefore necessarily fast evaluation of the created SPN models also requires further study. Currently we are experimenting with

[2]Even when using the statements `timer_Sleep()` or `instruct_Pause`.

[3]This `voucher/redeem` mechanism is described in [1].

MVA algorithms and the use of closed-form solutions for the SPNs [11].

The monitoring process is realized using two monitoring tools. In a future environment the use of one monitoring tool is preferred because of the interference of the monitoring process with the monitored applications. The current experience with generic monitoring shows satisfying results which makes it applicable for further use.

In this paper we mainly addressed pure performance issues. The use of replicated components and the evaluation of the models w.r.t. performability measures including dependability aspects, will be subject of further study. We also intend to use the performability manager as a conceptual framework for the study of resource control issues in multi-media conferencing systems.

References

[1] Architecture Projects Management Ltd. *ANSAware 4.0 application programmer's manual*, March 1992. Document RM.102.00.

[2] R.L. Bennett and G.E. Policello II. Switching Systems in the 21st Century. IEEE Communications Magazine: *Feature Topic: Toward The Global Intelligent Network*, 31(3):24–30, March 1993.

[3] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri Net Package. In *in Proc. 3rd Int. Workshop Petri Net and Performance Models, Kyoto, Japan*, pages 142–151. Duke University, Department of Computer Science, Durham, USA, IEEE Computer Society Press, December 1989.

[4] L.J.N. Franken and B.R.H.M. Haverkort. The Performability Manager. IEEE Network: The Magazine of Computer Communications *Special Issue on Distributed Systems for Telecommunications*, 8(1), Januari 1994.

[5] L.J.N. Franken, R.H. Pijpers, and B.R. Haverkort. Modelling Aspects of Model Based Dynamic QoS Management by the Performability Manager. *submitted to the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, May 1994.

[6] J.J. Garrahan, P.A. Russo, K. Kitami, and R. Kung. Intelligent Network Overview. IEEE Communications Magazine: *Feature Topic: Toward The Global Intelligent Network*, 31(3):30–38, March 1993.

[7] B.R. Haverkort and K.S. Trivedi. Specification and Generation of Markov Reward Models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

[8] H. Korte. Visualising ANSAware Programs with EXP93. Technical report, PTT Research, the Netherlands, unpublished, June 1993.

[9] F. Lange, R. Kroeger, and M. Gergeleit. JEWEL: Design and Implementation of a Distributed Measurement System. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):657–671, November 1992.

[10] MARI Computer Systems Ltd. *DEMON V3.0 User's guide and Reference manual*, 1993.

[11] R.H. Pijpers. Performability Monitoring and Modelling of ANSAware Environments. M.Sc. thesis, University of Twente, the Netherlands, December 1993.

[12] Studygroup XI. Q.1200, Draft recommendations. Technical report, CCITT, 1991.